**Universitatea Politehnica Bucureşti**

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Temă Sisteme de Operare

Device drivere și API pentru Android

**Studenți:**

**Petre Mihai Cătălin**

**Marcu Dragoș Alexandru**

**Prof. coordonator:**

**conf. dr. ing. Ştefan Stăncescu**

2015

**Cuprins**

[1) Introducere...................................................Petre Cătălin 3](#_Toc422169305)

[1.1) Ce sunt driverele? 3](#_Toc422169306)

[1.2) Ce este API? ...................................................Marcu Dragoș 4](#_Toc422169307)

[1.3) Despre Android 5](#_Toc422169308)

[2) Interfața USB...................................................Petre Cătălin 7](#_Toc422169309)

[2.1) Drivere USB 7](#_Toc422169310)

[2.1.1) Endpoint-uri 8](#_Toc422169311)

[2.1.2) USB Request Block (urb) 8](#_Toc422169312)

[2.1.3) Crearea unui driver USB 9](#_Toc422169313)

[2.2) API-uri USB...................................................Marcu Dragoș 12](#_Toc422169314)

[2.2.1) Cerințe Android Manifest 13](#_Toc422169315)

[2.2.2) Lucrul cu dispozitive 14](#_Toc422169316)

[2.2.3) Folosirea un filtru intenție 15](#_Toc422169317)

[3) Interfața Wi-Fi...................................................Petre Cătălin 16](#_Toc422169318)

[3.1) Notiuni de baza 16](#_Toc422169319)

[3.2) Crearea unei aplicatii Wi-Fi Direct folosind API 16](#_Toc422169320)

[3.2.1) Setari initiale: 16](#_Toc422169321)

[3.2.2) Detectarea conexiunilor: 17](#_Toc422169322)

[3.2.3) Conectarea la un punct 17](#_Toc422169323)

[3.2.4) Transferul datelor 17](#_Toc422169324)

[4) Bluetooth...................................................Marcu Dragoș 19](#_Toc422169325)

[4.1) API pentru Bluetooth 19](#_Toc422169326)

[4.2) Arhitectura 20](#_Toc422169327)

[5) Bibliografie 22](#_Toc422169328)

# Introducere

Scopul acestei lucrări este înțelegerea modului de funcționare al interfețelor dispozitivelor ce ruleaza sistemul de operare Android. Pentru ca interfețele unui dispozitiv mobil să fie folosite în interacțiunea cu alte dispozitive, este nevoie in primul rand de drivere.

## 1.1) Ce sunt driverele?

Aplicațiile pentru interfețele de conectare ale dispozitivelor mobile sunt reprezentate de drivere. Driverul este un program ce operează și/sau controlează o anumită interfață de conectare prezentă la un calculator sau la un dispozitiv mobil. Un driver reprezinta o interfață software ce permite sistemului de operare si a aplicatiilor sa acceseze functiile hardware-ului folosit.

“Driverele simplifica programarea actionand ca un translator între o componentă hardware și aplicatiile software folosite de sistemul de operare. De exemplu, la calculatoare, un driver pentru mouse trebuie sa accepte informații de la mouse, prin care se spune cat de mult s-a deplasat si ce buton a fost apasat.”[[1]](#endnote-1) (Tanenbaum, 2008)

În cazul calculatoarelor vorbim de drivere pentru doua categorii de echipamente: “ **echipamente bloc** (block devices), cum ar fi discurile, care conțin mai multe blocuri de date ce pot fi adresate separat si **echipamente caracter** (character devices), cum ar fi tastaturile si imprimantele, care genereaza sau accepta un flux de caractere” *[[2]](#endnote-2) (Tanenbaum, 2008).* În cazul dispozitivelor mobile vorbim însa de o singura categorie de drivere deoarece aceste dispozitive prezinta mai multe interfete decat un calculator. Pentru conectarea majoritatii echipamentelor la un dispozitiv mobil se foloseste în general interfața USB, insă interfața Wi-Fi este din ce in ce mai folosita pentru interactiunea cu diverse echipamente. De asemenea sunt folosite si alte interfete de tip wireless, cum ar fi Bluetooth sau, mai recent, NFC. Toate aceste interfețe au nevoie de drivere pentru a face legatura între echipamentele conectate și dispozitivul folosit. Driverele pentru mobile sunt, ca și in cazul calculatoarelor, specifice fiecărui sistem de operare, un driver pentru Android fiind incompatibil cu un dispozitiv cu iOS.

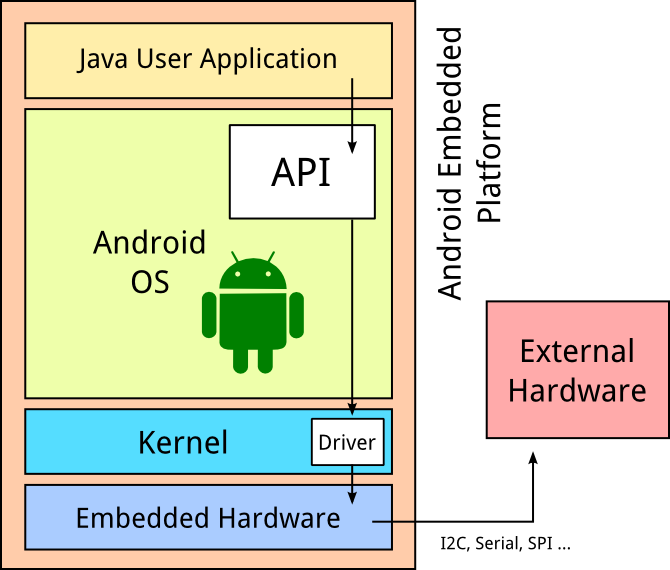
Device driverele folosesc API pentru a interactiona cu Kernel-ul si HAL-ul.

## 1.2) Ce este API?

API (Application Programming Interface) reprezintă o interfață pentru programarea aplicațiilor. API este un set de rutine, protocoale și unelte folosite pentru construirea aplicatiilor. Deci, API reprezintă defapt interfața dintre software si sistemul de operare.

În aceasta lucrare va fi prezentat modul în care o aplicație se foloseste de o interfață a dispozitivelor mobile ce ruleaza sistemul de operare Android (cel puțin versiunea 4.0). Majoritatea dispozitivelor de astăzi cu Android au numeroase interfețe hardware, de la cele clasice precum Wi-Fi, USB si Bluetooth, iar modelele cele mai performante includ si NFC.

Aplicațiile Android sunt scrise in limbajul de programare Java. Codul, alaturi de resurse și date, este compilat de Android SDK (Software Development Kit) și rezultă intr-un APK (Android Package). Sistemul de operare Android este un sistem multi-user Linux, în care fiecare aplicație reprezinta un user, sistemul asociind fiecarei aplicatii un Linux User ID unic. Fiecare aplicație rulează în propriul proces Linux.

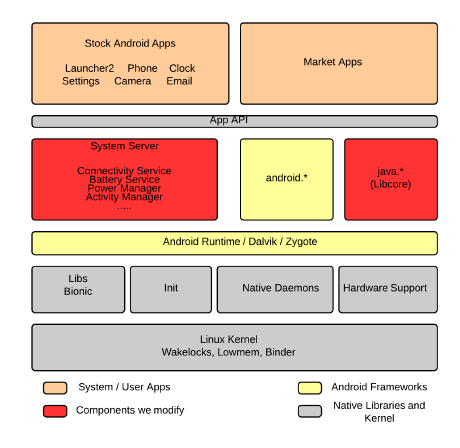
****

Figură 1 Situarea API in cadrul SO Android (sursa: http://www.fandroides.com/fandroides-expert-que-es-una-api/)

## 1.3) Despre Android

Android este cel mai popular sistem de operare pentru dispozitivele mobile cu ecran tactil, cum ar fi telefoanele inteligente și tabletele, însa este în plină dezvoltare și pentru televiziune (Android TV), mașini (Android Auto) și ceasuri inteligente (Android Wear). A fost creat de Google in anul 2008 si in momentul de fața este cel mai folosit sistem de operare de orice tip.*[[3]](#endnote-3) (Manjoo, 2015)*

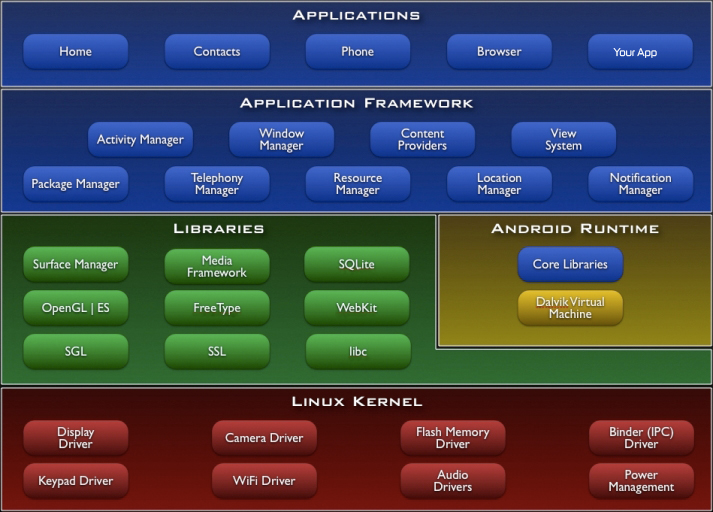
Este de tip open-source și astfel este folosit de mulți dezvoltatori pentru a modifica funcționalitățile existente sau adăuga unele noi. În Figurile 2 și 3 se prezintă arhitectura sistemului Android.



Figură 2 Arhitectura Android (sursa: Varun Anand, Multi-Interface Connectivity on Android, 2014)

Android se bazează pe kernel-ul Linux, cu anumite modificari pentru a se adapta la dispozitivele mobile, cum ar fi Binder IPC si lowmem killer. Arhitectura Android consta in biblioteci separate ce au numeroase functionalitati. In centrul arhitecturii se gaseste masina virtuala Dalvik care este esentiala pentru toate aplicatiile. Aplicatiile pentru Android sunt scrise in Java si apoi sunt compilate in bytecode-ul dalvik pentru a fi interpretate de catre masina virtuala. Biblioteca System Server interactioneaza cu interfetele disponibile pe un dispozitiv mobil, le gestioneaza, si ofera informatii despre acestea catre aplicatii.

Deasupra kernel-ului Linux, se gasesc biblioteci, middleware si API-uri scrise in limbajul C, si aplicatii software ce ruleaza pe un cadru de aplicatii care include biblioteci Java-compatibile bazate pe Apache Harmony.[[4]](#endnote-4) (Anand, 2014)



Figură 3 Diagrama arhitecturii Android (sursa: https://en.wikipedia.org/wiki/Android\_%28operating\_system%29)

In Figura 3 sunt prezentate toate componentele sistemului Android, organizate pe niveluri. Asa cum se poate observa si in aceasta diagrama, functionalitatea sistemului de operare se bazeaza pe drivere, care se regasesc in Linux kernel.

*Kernel*-ul este un program ce cuprinde toate driverele dispozitivului care conecteaza aplicatiile software la hardware.

In *biblioteci* se gasesc toate codurile care se pot folosi pentru crearea diverselor aplicatii in Android: OpenGL, WebKit Library, SQLite.

*Android Runtime* contine bibliotecile de baza si Masina Virtuala Dalvik. Dalvik VM este asemanatoare cu Java VM cu exceptia faptului ca este optimizata sa ruleze aplicatii Android si este de asemenea, responsabila pentru managementul memoriei, multi-threading.

*Application Framework* cuprinde un grup de API-uri care se folosesc pentru dezvoltarea aplicatiilor. Prin API se administreaza modul in care aplicatiile interactioneaza cu sistemul.

*Interfata HAL*

HAL (Hardware Abstaction Layer) este un subsistem al sistemelor de operare format din rutine ce permit programelor accesul la resursele hardware. HAL este apelat de catre Kernel prin intermediul driverelor unor anumite componente, gasindu-se intre framework-ul Android si software-ul specific interactiunii cu hardware-ul.

# Interfața USB

## 2.1) Drivere USB

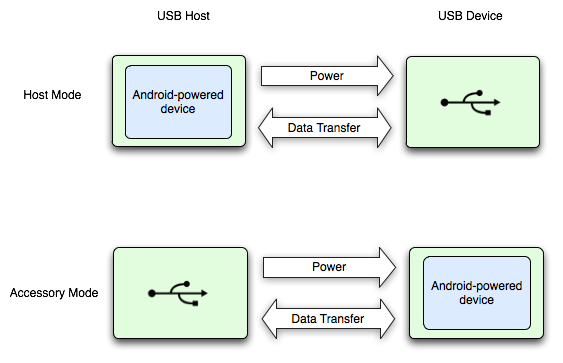
Interfata USB este destul de complexa, insa kernel-ul Android, care se bazeaza pe kernel-ul Linux, ofera un subsistem numit *USB core* pentru a rezolva complexitatea acesteia.

Kernel-ul Android suporta doua tipuri de drivere USB: drivere de tip **host** (pe sistemul gazda) si de tip **dispozitiv** (accesoriu).

Driverele USB pentru host controleaza dispozitivele USB controleaza dispozitivul USB care este conectat la gazda, iar driverele pentru dispozitiv controleaza dispozitivul care se conecteaza la un computer, de exemplu.

În modul accesoriu USB, hardware-ul USB extern se comporta ca o gazdă USB. Cateva accesorii ce se pot conecta la USB includ controlere de roboți, stații de andocare, echipamente de muzicale, echipamente de diagnosticare, cititoare de carduri și multe altele. Aceastea dau dispozitivelor ce ruleaza Android dar care nu au capacități de gazda, capacitatea de a interacționa cu hardware-ul USB . Accesoriile USB Android trebuie să fie proiectate pentru a lucra cu dispozitive Android și trebuie să adere la *protocolul de comunicare cu accesorii Android*. În modul USB gazdă, dispozitivul Android acționează în calitate de gazdă . Exemple de astfel de dispozitive includ camere digitale, tastaturi, mouse-uri și controlere de jocuri.

In Figura 3 sunt prezentate diferentele dintre aceste doua moduri. Cand dispozitivul Android este in modul gazda, actioneaza ca un USB host si deschide bus-ul. Cand dispozitivul Android este in modul accesoriu, hardware-ul conectat prin USB actioneaza precum o gazda si deschide bus-ul. [[5]](#endnote-5)



Figură 4 Modurile USB Host si USB Accesoriu (sursa: https://developer.android.com/guide/topics/connectivity/usb/index.html)

### 2.1.1) Endpoint-uri

Cea mai de baza forma de comunicatie prin USB se produce prin *end-point*. End-point-urile sunt structuri de date prin care se transporta datele dintr-o directie in alta, pe canalul de comunicatie dintre dispozitive (ex.: registre I/O, memorii tampon locale). Un end-point USB poate transporta datele fie de la host catre accesoriu (denumit OUT endpoint), fie de la accesoriu catre host (denumit IN endpoint). Exista patru tipuri de endpoint-uri USB:

*Control* – aceste endpointuri sunt folosite pentru a permite accesul la diferite parti ale dispozitivului USB. Sunt folosite in general pentru configurarea dispozitivului, preluarea informatiei despre dispozitiv, trimiterea comenzilor catre dispozitiv.

*Interrupt* (intrerupere) – aceste endpoint-uri transfera cantitati mici de date la o viteza fixa de fiecare data cand host-ul USB cere date de la dispozitiv.

*Bulk* – endpoint-urile bulk transfera cantitati mari de date, mult mai mari decat in cazul endpoint-urilor interrupt. Sunt intalnite la dispozitivele care trebuie sa transmita date de dimensiuni mari fara pierderi.

*Isochronous* – aceste endpointuri transfera de asemenea cantitati mari de date, insa nu este garantata transmiterea fara pierderi; pot fi folosite de catre dispozitive care nu sunt afectate de pierderile de date.

Endpoint-urile USB sunt descrise in Kernel cu structura

*struct usb\_host\_endpoint* iar informatia reala despre aceasta structura se gaseste in *struct usb\_endpoint\_descriptor*.

Driverele sunt interesate de urmatoarele campuri ale structurii:

*bEndpointAddress* – aici se specifica daca datele trebuie directionata catre dispozitiv sau catre host (USB\_DIR\_OUT si USB\_DIR\_IN)

*bmAttributes* – reprezinta tipul endpoint-ului

*wMaxPacketSize* – dimensiunea maxima in biti pe care endpoint-ul o poate sustine

*bInterval* – aceasta valoare reprezinta intervalul dintre cererile interrupt pentru endpoint

### 2.1.2) USB Request Block (urb)

Codul USB din kernel interactioneaza cu dispozitivele USB prin cereri bloc ale USB-ului denumite *urb (USB request block)*. Un urb este folosit pentru a primi date sau trimite date, de la sau catre un endpoint USB dintr-un dispozitiv USB. Un driver USB poate aloca mai multe urb pentru un singur endpoint sau poate refolosi acelasi urb in mai multe endpoint-uri, in functie de necesitatile driverului. Un ciclu de viata al unui urb este urmatorul:

* Este creat de driverul de dispozitiv USB
* Este asociat unui anumit endpoint al unui driver USB
* Trimis catre USB Core, de catre driver
* Trimis de USB Core catre driverul de control de hosturi USB al dispozitivului
* Procesat de catre driverul de control care face transferul USB catre dispozitiv
* Cand cererea urb este completa, driverul de control informeaza driverul de dispozitiv USB

*Structura unui urb*

* *struct usb\_device \*dev* - indicarea dispozitivului catre care se trasmite urb-ul
* *unsigned int pipe* - informația despre endpoint-ul dispozitivului catre care se transmite urb-ul. Aceasta variabila trebuie initializata de driverul USB inainte ca urb-ul sa fie trimis catre core-ul USB
* *int status* – aceasta variabila este setata in functie de starea curenta a cererii bloc (daca a fost trimisa sau este in curs de procesare)
* *unsigned int transfer\_flags* – variabila setata in functie de ce doreste driverul USB sa se intample cu urb-ul
* *void \* transfer\_buffer* – pointer catre buffer pentru a fi folosit cand se trimit date catre dispozitiv sau cand se primesc date de la dispozitiv
* *dma\_addr\_t transfer\_dma* – buffer ce este folosit la transferul de date catre dispozitivul USB folosind DMA
* *int transfer\_buffer\_length* – lungimea bufferului folosit la transfer
* *unsigned char \*setup\_packet* – pointer catre pachetul de setup pentru un urb de control
* *dma\_addr\_t setup\_dma* – bufferul DMA pentru pachetul de setup
* *int start\_frame* – setează sau returnează numărul frame-ului initial
* *struct usb\_iso\_packet\_descriptor  iso\_frame\_desc[0]* – valabil doar pentru urb-uri sincronice; permite unui singur urb sa defineasca un numar de transferuri sincronice in acelasi moment

### 2.1.3) Crearea unui driver USB

Structura principala pe care driverele USB trebuie sa o creeze este

*struct usb\_driver.* Aceasta structura trebuie sa contina functii si variabile care descriu driverul USB catre core-ul USB:

* *struct module \*owner* – indicator catre detinatorul modulului driverului respectiv
* *const char \*name* – indicator catre numele driverului; trebuie sa fie unic intrea toate driverele USB din kernel
* *const struct usb\_device\_id \*id\_table* – indicator catre tabela *struct usb\_device\_id*
* *int (\*probe) (struct usb\_interface \*intf, const struct usb\_device\_id \*id)* – indicator catre functia driverului USB
* *void (\*disconnect) (struct usb\_interface \*intf)* – indicator catre functia de deconectare a driverului

Deci, pentru crearea unei structuri usb\_driver, trebuiesc initializate doar cinci campuri:

static struct usb\_driver skel\_driver = {

.owner = THIS\_MODULE,

.name = "skeleton",

.id\_table = skel\_table,

.probe = skel\_probe,

.disconnect = skel\_disconnect,

};

Structura usb\_driver mai poate contine cateva apeluri, dar acestea nu sunt folosite prea des si nu sunt necesare pentru ca un driver USB sa functioneze corect.

Cand driverul urmeaza sa fie descarcat, struct usb\_driver trebuie sa dispara din kernel. Acest lucru este realizat de apelul usb\_deregister\_driver: interfata USB care apartine driverului este deconectata prin functia *disconnect.*

static void \_\_exit usb\_skel\_exit(void)

{

/\* deregister this driver with the USB subsystem \*/

usb\_deregister(&skel\_driver);

}

Transmiterea si controlul unui urb:

Cand driverul are de trimis date catre dispozitivul USB, trebuie alocat un urb pentru transmiterea datelor.

urb = usb\_alloc\_urb(0, GFP\_KERNEL);

if (!urb) {

retval = -ENOMEM;

goto error;

}

Dupa ce urb-ul a fost alocat cu succes, un buffer DMA trebuie creat pentru a trimite datele catre dispozitiv intr-un mod cat mai eficient, iar datele care trebuie trimise catre driver trebuie copiate in acest buffer:

buf = usb\_buffer\_alloc(dev->udev, count, GFP\_KERNEL, &urb->transfer\_dma);

if (!buf) {

retval = -ENOMEM;

goto error;

}

if (copy\_from\_user(buf, user\_buffer, count)) {

retval = -EFAULT;

goto error;}

Odata ce datele au fost copiate cu succes in buffer, trebuie initializat un urb inainte de a fi trimis catre core-ul USB:

/\* initialize the urb properly \*/

usb\_fill\_bulk\_urb(urb, dev->udev,

usb\_sndbulkpipe(dev->udev, dev->bulk\_out\_endpointAddr),

buf, count, skel\_write\_bulk\_callback, dev);

urb->transfer\_flags |= URB\_NO\_TRANSFER\_DMA\_MAP;

Dupa ce urb-ul a fost alocat in mod corespunzator, datele au fost copiate corect, si urb-ul este initializat corect, poate fi prezentat catre core-ul USB pentru a fi transmis catre dispozitiv:

/\* send the data out the bulk port \*/

retval = usb\_submit\_urb(urb, GFP\_KERNEL);

if (retval) {

err("%s - failed submitting write urb, error %d", \_\_FUNCTION\_\_, retval);

goto error;

}

Daca urb-ul este transmis cu succes catre dispozitiv, este cerut raspunsul urb-ului de catre core-ul USB. In urmatorul exemplu, a fost initializat urb-ul pentru a indica spre functia *skel\_write\_bulk\_callback.*

static void skel\_write\_bulk\_callback(struct urb \*urb, struct pt\_regs \*regs)

{

/\* sync/async unlink faults aren't errors \*/

if (urb->status &&

!(urb->status = = -ENOENT ||

urb->status = = -ECONNRESET ||

urb->status = = -ESHUTDOWN)) {

dbg("%s - nonzero write bulk status received: %d",

\_\_FUNCTION\_\_, urb->status);

}

/\* free up our allocated buffer \*/

usb\_buffer\_free(urb->dev, urb->transfer\_buffer\_length,

urb->transfer\_buffer, urb->transfer\_dma);

}

Primul lucru pe care il face functia de raspuns este sa verifice starea urb-ului pentru a determina daca cererea este completata cu succes sau nu. Apoi raspunsul elibereaza buffer-ul alocat ce a fost asociat acestui urb in timpul transmiterii.

*[[6]](#endnote-6) (Corbet, Rubini, & Kroah-Hartman, 2005)*

## 2.2) API-uri USB

Următorul tabel descrie API-urile gazdă USB din pachetul android.hardware.usb.

|  |  |
| --- | --- |
| Usb Manager | Permite comunicarea cu dispozitivele USB conectate . |
| Usb Device | Reprezintă un dispozitiv USB conectat și conține metode de a accesa de informații de identificare , interfețe , și obiective . |
| Interfata USB | Reprezintă o interfață a unui dispozitiv USB , care definește un set de funcționalități pentru dispozitiv . Un dispozitiv poate avea una sau mai multe interfețe pe care să comunice mai departe. |
| USB Endpoint | Reprezintă un obiectiv interfață , care este un canal de comunicare pentru această interfață . O interfata poate avea unul sau mai multe obiective , și, de obicei are intrare și ieșire obiective de comunicare în ambele sensuri cu dispozitivul . |
| Usb Device Connection | Reprezintă o conexiune cu dispozitivul , care transferă datele pe obiective. Aceasta clasa vă permite să trimiteți datele înainte și înapoi sincran sau asincron . |
| UsbRequest | Reprezintă o cerere asincron pentru a comunica cu un dispozitiv prin intermediul unui dispozitiv de conexiune USB(UsbDeviceConnection). |
| Constante USB | Conține constante pentru protocolul USB . |

În cele mai multe situații , trebuie să se folosească toate aceste clase

(UsbRequest este necesară numai în cazul în care se face comunicare asincronă), atunci când comunică cu un dispozitiv USB . În general, se obtine un Manager USB pentru a prelua dispozitivul USB dorit. La dispozitiv, este nevoie sa se gaseasca Interfata USB corespunzătoare și punctul final USB al interfeței pentru a comunica mai departe. Odată ce s-a obținut punctul final corect, se deschide o conexiune pentru a comunica cu dispozitivul USB.

### 2.2.1) Cerințe Android Manifest

Următoarea listă descrie ceea ce trebuie sa contina fișierul manifest înainte de lucrul cu API-urile gazdă USB:

Deoarece nu toate dispozitivele Android sunt garantate pentru a suporta API-urile gazdă USB, este inclus un element <uses-feature> care declară că cererea utilizatorului foloseste caracteristica android.hardware.usb.host.

Trebuie setata valoarea SDK minimă a cererii de API de nivel 12 sau mai mare. API-urile gazdă USB nu sunt prezente pe niveluri API anterioare.

Dacă se doreste ca cererea să fie notificată de un dispozitiv USB atașat, trebuie specificat un <intent-filter> și <meta-data>, elemente pereche utilizate pentru android.hardware.usb.action, USB\_DEVICE\_ATTACHED în activitatea principală. <meta-data> reprezinta fișierele externe de resurse XML care declară informații de identificare despre dispozitivul care trebuie detectat

În fișierul de resurse XML, trebuie declarate in <usb-device> elementele pentru dispozitivele USB care trebuie filtrate. Următoarea listă descrie atributele <usb-device>. În general, se utilizeaza vanzatorul și ID-ul produsului dacă se face filtrarea pentru un anumit dispozitiv și utilizarea de clasă si subclasă, iar protocolul dacă se face filtrarea pentru un grup de dispozitive USB, cum ar fi dispozitive de stocare în masă sau camere digitale. Se pot specifica toate aceste atribute sau niciunul. Specificarea niciunui atribut corespunde fiecarui dispozitiv USB, iar acest lucru se utilizeaza daca cererea necesita:

-furnizor-id

-produs-id

-clasă

-subclasă

-protocol (dispozitiv sau interfata)

Se salveaza fișierul de resurse în res / xml / director. Numele fișierului de resurse (fără extensia .xml) trebuie să fie de același fel cu cel specificat în <meta-data> element. Formatul de fișier XML de resurse este in exemplul de mai jos.

Următorul exemplu arată o mostră manifest și dosarul său de resurse corespunzătoare :

<manifest ...>  
    <uses-feature android:name="android.hardware.usb.host" />  
    <uses-sdk android:minSdkVersion="12" />  
    ...  
    <application>  
        <activity ...>  
            ...  
            <intent-filter>  
                <action android:name="android.hardware.usb.action.USB\_DEVICE\_ATTACHED" />  
            </intent-filter>  
  
            <meta-data android:name="android.hardware.usb.action.USB\_DEVICE\_ATTACHED"  
                android:resource="@xml/device\_filter" />  
        </activity>  
    </application>  
</manifest>

În acest caz , următorul fișier de resurse ar trebui să fie salvat în res / xml / device\_filter.xml, cu precizarea că orice dispozitiv USB cu atributele specificate ar trebui filtrate :

<?xml version="1.0" encoding="utf-8"?>  
  
<resources>  
    <usb-device vendor-id="1234" product-id="5678" class="255" subclass="66" protocol="1" />  
</resources>

### 2.2.2) Lucrul cu dispozitive

Când utilizatorii conecteaza dispozitive USB la un dispozitiv Android, sistemul Android poate determina dacă cererea este interesata de dispozitivul conectat. Dacă da, exista posibilitatea să se configureze comunicarea cu dispozitivul. Pentru a face acest lucru, aplicația trebuie să:

-Descoperirea dispozitivelor USB conectate prin utilizarea unui filtru de intenție pentru a primi o notificare atunci când utilizatorul se conectează la un dispozitiv USB sau prin enumerarea dispozitivelor USB care sunt deja conectate.

Utilizatorului i se cere permisiunea să se conecteze la dispozitivul USB, dacă nu este deja obținută.

Comunicarea cu dispozitivul USB se face prin citirea și scrierea datelor pe obiectivele de interfață corespunzătoare.

Descoperirea unui dispozitiv

- Aplicația poate descoperi dispozitive USB fie prin utilizarea unui filtru intenție pentru a se notifica atunci când utilizatorul se conectează un dispozitiv fie prin enumerarea dispozitivelor USB care sunt deja conectate. Folosirea unui filtru intentie este utila dacă se doreste ca aplicația să detecteze automat un dispozitiv dorit. Enumerarea dispozitivele USB conectate este utila dacă se doreste obtinerea unei liste cu toate dispozitivele conectate.

Dacă cererea facuta este interesată în inspectarea tuturor dispozitivelor USB conectate în prezent în timp ce cererea se execută , poate enumera dispozitivele . Utilizand metoda getDeviceList ( ) se obține o hartă (hash) cu toate dispozitivele USB care sunt conectate . Harta hash este fixată de numele dispozitivului USB , dacă doriți pentru a obține un dispozitiv de pe hartă .

UsbManager manager = (UsbManager) getSystemService(Context.USB\_SERVICE);  
...    
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();  
UsbDevice device = deviceList.get("deviceName");

### 2.2.3) Folosirea un filtru intenție

Pentru ca aplicația să descopere un anumit dispozitiv USB , puteți specifica un filtru de intenție android.hardware.usb.action.USB\_DEVICE\_ATTACHED . Odata cu acest filtru de intenție , trebuie să specificați un fișier de resurse care specifică proprietățile dispozitivului USB , cum ar fi produse și furnizor de identitate. Când utilizatorii conectați la un dispozitiv care se potrivesc filtrului aparat , sistemul le prezintă cu un dialog care întreabă dacă vor să înceapă aplicația . Dacă utilizatorii acceptă , aplicația are automat permisiunea de a accesa dispozitivul până când dispozitivul este deconectat .

Următorul exemplu arată cum se declară filtru intenție :

<activity ...>  
...  
    <intent-filter>  
        <action android:name="android.hardware.usb.action.USB\_DEVICE\_ATTACHED" />  
    </intent-filter>  
  
    <meta-data android:name="android.hardware.usb.action.USB\_DEVICE\_ATTACHED"  
        android:resource="@xml/device\_filter" />  
</activity>

Următorul exemplu arată cum se declară fișierul de resurse corespunzătoare, care să specifice dispozitivele USB care vă interesează:

<?xml version="1.0" encoding="utf-8"?>  
  
<resources>  
    <usb-device vendor-id="1234" product-id="5678" />  
</resources>

# Interfața Wi-Fi

## 3.1) Notiuni de baza

Wi-Fi este o tehnologie de conectare in reteaua locala wireless ce are rolul de a realiza conexiunea dispozitivelor electronice la retea, utilizand frecvente radio.

Wi-Fi Direct, sau P2P (Peer-to-Peer), este un standard Wi-Fi ce permite conexiunea intre doua sau mai multe dispozitive, de tipul punct la punct (peer to peer) fara a fi necesar un Wireless Access Point. Poate fi folosit de la activitati precum browsing-ul pe internet pana la transferul de fisiere si la comunicarea cu mai multe dispozitive la viteze Wi-Fi normale. Avantajul Wi-Fi Direct este capacitatea de a conecta dispozitive ce apartin de producatori diferiti.

Folosind aceste API-uri, un dispozitiv detecteaza si se conecteaza la un altul ce suporta Wi-Fi P2P, apoi comunica intre ele pe distante mult mai mari decat intr-o conexiune Bluetooth.[[7]](#endnote-7)

## 3.2) Crearea unei aplicatii Wi-Fi Direct folosind API

Crearea unei aplicatii Wi-Fi P2P implica crearea si inregistrarea unui receiver broadcast pentru aplicatie, descoperirea utilizatorilor, conectarea la un utilizator, si transferul de date catre un utilizator.

### 3.2.1) Setari initiale:

Pasul 1) Cererea permisiunii de a utiliza hardware-ul Wi-Fi de pe dispozitiv si verificarea versiunii corecte de SDK se face folosind urmatoarele comenzi:

<uses-sdk android:minSdkVersion="14" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

Pasul 2) Verificarea daca modul Wi-Fi P2P este pornit si functioneaza. Acest lucru se poate verifica in cadrul receiverului broadcast cand acesta primeste apelul WIFI\_P2P\_STATE\_CHANGED\_ACTION

@Override  
public void onReceive(Context context, Intent intent) {  
    ...  
    String action = intent.getAction();  
    if (WifiP2pManager.WIFI\_P2P\_STATE\_CHANGED\_ACTION.equals(action)) {  
       int state = intent.getIntExtra(WifiP2pManager.EXTRA\_WIFI\_STATE, -1);  
       if (state == WifiP2pManager.WIFI\_P2P\_STATE\_ENABLED) {  
            // Wifi P2P este activat  
        } else {  
            // Wi-Fi P2P nu este activat  
        }  
    }  
    ...  
}

Pasul 3) In modul onCreate() se realizeaza majoritatea initializarilor. Prin aceasta metoda se creaza instante cereturneaza un manager (WifiP2pManager.Channel) care este folosit pentru conectarea aplicatiei in cadrul Wi-Fi. Dupa crearea acestor instante, se permite receptorului broadcast sa observe modificari ale conectivitatii.

Pasul 4) Se creaza un filtru de intentie in care se adauga aceleasi intentii pe care receptorul broadcast le verifica.

mIntentFilter.addAction(WifiP2pManager.WIFI\_P2P\_STATE\_CHANGED\_ACTION);  
    mIntentFilter.addAction(WifiP2pManager.WIFI\_P2P\_PEERS\_CHANGED\_ACTION);  
    mIntentFilter.addAction(WifiP2pManager.WIFI\_P2P\_CONNECTION\_CHANGED\_ACTION);  
    mIntentFilter.addAction(WifiP2pManager.WIFI\_P2P\_THIS\_DEVICE\_CHANGED\_ACTION);

Pasul 5) Se inregistreaza receiver-ul broadcast in metoda onResume() si se retrage din onPause()

### 3.2.2) Detectarea conexiunilor:

Pentru descoperirea conexiunilor disponibile, se apeleaza discoverPeers() pentru a detecta conexiunile din apropiere. Apelul la aceasta functie este asincron si succesul sau esecul la conectare este semnalizat in managerul Wi-Fi P2P creat mai devreme.

Daca procesul de detectare a conexiunilor are succes, sistemul transmite un mesaj broadcast catre modulul WIFI\_P2P\_PEERS\_CHANGED\_ACTION, dupa care se poate cere o lista afisata a conexiunilor disponibile prin requestPeers().

### 3.2.3) Conectarea la un punct

Dupa ce se obtine lista conexiunilor disponibile si se cunoaste dispozitivul la care se doreste conectarea, se apeleaza metoda connect(), iar datele despre dispozitivul la care se va conecta sunt continute in WifiP2pConfig.

### 3.2.4) Transferul datelor

Odata ce conexiunea a fost stabilita, se pot transfera date intre dispozitivele conectate. Pasii de baza in transferul de date sunt:

* Crearea unui ServerSocket, ce asteapta o conexiune de la un client pe un anumit port pe care il blocheaza pana se face conectarea
* Crearea unui client Socket, ce folosete IP-ul si portul socket-ului server pentru a se conecta la dispozitivul server
* Transmiterea datelor de la client catre server
* Socket-ul server asteapta conexiunea unui client ce primeste datele

In exemplul urmator se realizeaza transmiterea unui fisier JPEG in sistemul de fisiere al dispozitivul client.

Context context = this.getApplicationContext();  
String host;  
int port;  
int len;  
Socket socket = new Socket();  
byte buf[]  = new byte[1024];  
...  
try {  
    /\*\*  
     \* Crearea unui socket client cu informatii despre host,  
     \* port si timeout.  
     \*/  
    socket.bind(null);  
    socket.connect((new InetSocketAddress(host, port)), 500);  
    /\*\*  
     \* Crearea unui flux de octet de la un fisier JPEG si transportarea lui la un flux de iesire din socket.   
     \*. Aceste date vor fi intoarse de catre dispozitivul server.  
     \*/  
    OutputStream outputStream = socket.getOutputStream();  
    ContentResolver cr = context.getContentResolver();  
    InputStream inputStream = null;  
    inputStream = cr.openInputStream(Uri.parse("path/to/picture.jpg"));  
    while ((len = inputStream.read(buf)) != -1) {  
        outputStream.write(buf, 0, len);  
    }  
    outputStream.close();  
    inputStream.close();  
} catch (FileNotFoundException e) {  
    //catch logic  
} catch (IOException e) {  
    //catch logic  
}  
/\*\*  
 \* Inchiderea tuturor socketurilor deschise cand se termina transferul  
 \* sau daca a aparut o exceptie.  
 \*/  
finally {  
    if (socket != null) {  
        if (socket.isConnected()) {  
            try {  
                socket.close();  
            } catch (IOException e) {  
                //catch logic  
            }  
        }  
    }  
}

# Bluetooth

## 4.1) API pentru Bluetooth

Bluetooth este o tehnologie wireless utilizata pentru schimbul de date pe distante scurte intre dispozitive mobile si fixe, dar si pentru crearea de retele personale (personal area networks – PAN).

Platforma Android include suport pentru stiva de rețea Bluetooth, care permite unui dispozitiv sa faca schimb de date fără fir cu alte dispozitive Bluetooth. Cadrul aplicație oferă acces la funcționalitatea Bluetooth prin Android Bluetooth API. Aceste API-uri lasa aplicații conecta fără fir la alte dispozitive Bluetooth, care permite punct la punct și multipunct caracteristici fără fir.

Folosind API Bluetooth, o aplicatie Android poate:

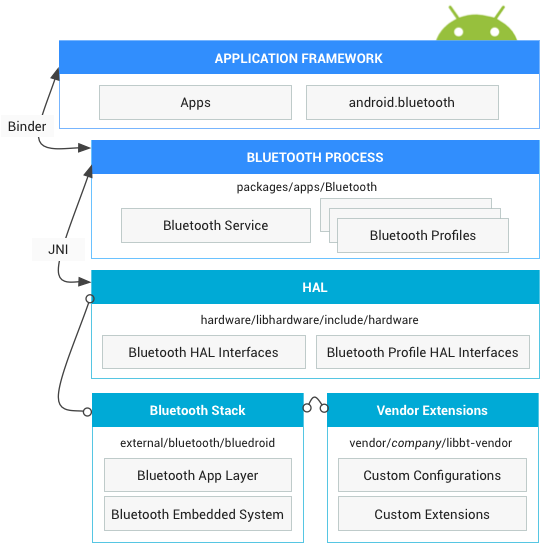
* Scana pentru alte dispozitive Bluetooth
* Interoga adaptorul locale Bluetooth pentru dispozitive Bluetooth asociate
* Stabili canale RFCOMM
* Conecta la alte dispozitive prin descoperire de servicii
* Transfera date către și de la alte dispozitive
* Gestiona mai multe conexiuni

Classic Bluetooth este alegerea potrivita pentru mai multe operații mari consumatoare de baterie, cum ar fi streaming-ul și comunicarea între dispozitive Android. Dispozitivele Bluetooth cu cerințe reduse de energie introduc suport API pentru Bluetooth Low Energy.

Android oferă implicit Stiva Bluetooth, BlueDroid, care este împărțita în două straturi: sistemul Bluetooth încorporat (BTE) , care pune în aplicare funcționalitatea pe bază Bluetooth și stratul Bluetooth Application (BTA) , care comunică cu aplicatiile in cadrul Android.

## 4.2) Arhitectura

Un serviciu de sistem Bluetooth comunică cu stiva Bluetooth prin JNI și cu aplicații prin Binder IPC . Serviciul de sistem oferă dezvoltatorilor acces la diverse profiluri Bluetooth . Următoarea diagramă prezintă structura generală a stivei Bluetooth :



Figură 5: Stiva Bluetooth (Sursa https://source.android.com/devices/bluetooth.html)

Stiva Bluetooth

Implicit stiva Bluetooth este oferită pentru utilizator. Stiva implementează generic Bluetooth HAL , precum se particularizează cu extensii și modificări de configurare .

Framework-ul aplicatiei

La nivelul framework-ului de aplicare este codul aplicației , care utilizează API-urile android.bluetooth să interacționeze cu hardware-ul Bluetooth . Pe plan intern, acest cod apeleaza procesul de Bluetooth prin mecanismul Binder IPC .

Serviciu de sistem Bluetooth

Serviciul de sistem Bluetooth, situat în pachete/apps/Bluetooth, este ambalat ca o aplicatie Android și implementează serviciul Bluetooth și profilurile la framework-ul Android . Această aplicație se pune în stratul HAL prin JNI .

Extensii de furnizor

Pentru a adăuga extensii personalizate și un strat HCI pentru urmărire, se poate crea un modul libbt - furnizor și să se precizeze aceste componente .

JNI

Codul JNI asociat android.bluetooth este situat în pachete/ apps/Bluetooth/JNI . Codul JNI cheamă în stratul HAL și primește callback de la HAL atunci când apar anumite operațiuni Bluetooth, cum ar fi atunci când dispozitivele sunt descoperite .

HAL

Nivelul de abstractizare hardware defineste interfata standard în care API-urile *android.bluetooth* și procesul Bluetooth sunt puse și pe care trebuie să le pună în aplicare în mod corect pentru a avea funcția de hardware Bluetooth. Fișierele antet pentru HAL Bluetooth se află în

*hardware / libhardware / include / hardware / bluetooth.h și*

*hardware / libhardware / include / hardware / bt \_ \* . Fișiere h .*

Punerea în aplicare a HAL

Bluetooth HAL este situat în hardware/libhardware/include/hardware/ directorul:

*bluetooth.h:* Include definiția interfață pentru hardware-ul Bluetooth de pe dispozitiv.

*bt\_av.h:* Include definiția interfeței pentru profilul A2DP.

*bt\_gatt.h, bt\_gatt\_client.h și bt\_gatt\_server.h*: Acestea includ definirea interfeței pentru profilul GATT.

*bt\_hf.h*: Include definiția interfeței pentru profilul HFP.

*bt\_hh.h:* Include definiția interfeței pentru profilul HID gazdă.

*bt\_hl.h*: Include definiția interfeței pentru profilul HDP.

*bt\_mce.h*: Include definiția interfeței pentru profilul MAP.

*bt\_pan.h*: Include definiția interfeței pentru profilul PAN.

*bt\_rc.h:* Include definiția interfeței pentru profilul AVRCP.

*bt\_sock.h*: Include definiția interfata pentru prize RFCOMM.

Punerea în aplicare Bluetooth nu este constrânsă de caracteristicile și profilurile expuse în HAL. Se pot găsi aplicatii implicite situate în stiva BlueDroid Bluetooth in bluetooth/directorul extern/bluedroid, care implementează implicit HAL și, de asemenea, caracteristici suplimentare și particularizări. [[8]](#endnote-8)

# Bibliografie

Andrew Tanenbaum, „**Modern Operating Systems”,** 3'rd edition, Pearson Education Inc., 2008

Farhad Manjoo, „**A Murky Road Ahead for Android**”, 2015, *The New York Times*

Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. **„Linux Device Drivers 3rd Edition”**, 2005, O'Reilly Media.

Varun Anand, „**Multi-Interface Connectivity on Android**”, 2014

http://en.wikipedia.org/wiki/Application\_programming\_interface

http://en.wikipedia.org/wiki/Device\_driver#APIs

http://developer.android.com/guide/topics/connectivity/wifip2p.html

http://developer.android.com/guide/topics/connectivity/usb/index.html

https://source.android.com/devices/bluetooth.html

**Referințe**

1. Tanenbaum, A. S. (2008). Modern *Operating Systems 3'rd edition.* Pearson Education Inc. [↑](#endnote-ref-1)
2. Tanenbaum, A. S. (2008). Modern *Operating Systems 3'rd edition.* Pearson Education Inc. [↑](#endnote-ref-2)
3. Manjoo, Farhad (2015-05-27). "A Murky Road Ahead for Android, Despite Market Dominance". *The New York Times* [↑](#endnote-ref-3)
4. Varun Anand, „Multi-Interface Connectivity on Android”, 2014 [↑](#endnote-ref-4)
5. http://developer.android.com/guide/topics/connectivity/usb/index.html [↑](#endnote-ref-5)
6. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman (2005). Linux Device Drivers 3rd Edition. O'Reilly Media. Capitolul 13 – Drivere USB [↑](#endnote-ref-6)
7. http://developer.android.com/guide/topics/connectivity/wifip2p.html [↑](#endnote-ref-7)
8. https://source.android.com/devices/bluetooth.html [↑](#endnote-ref-8)